**Que: Find out the Linux kernel development process?**

**Ans:** Linux kernel development in the early 1990’s was a pretty loose affair, with relatively small numbers of users and developers involved. With a user base in the millions and with some 2,000 developers involved over the course of one year, the kernel has since had to evolve a number of processes to keep development happening smoothly. A solid understanding of how the process works is required in order to be an effective part of it.

**The big picture:**

The kernel developers use a loosely time-based release process, with a new major kernel release happening every two or three months. The recent release history looks like this:

|  |  |
| --- | --- |
| 2.6.38 | March 14, 2011 |
| 2.6.37 | January 4, 2011 |
| 2.6.36 | October 20, 2010 |
| 2.6.35 | August 1, 2010 |
| 2.6.34 | May 15, 2010 |
| 2.6.33 | February 24, 2010 |

Every 2.6.x release is a major kernel release with new features, internal API changes, and more. A typical 2.6 release can contain nearly 10,000 change sets with changes to several hundred thousand lines of code. 2.6 is thus the leading edge of Linux kernel development; the kernel uses a rolling development model which is continually integrating major changes.

A relatively straightforward discipline is followed with regard to the merging of patches for each release. At the beginning of each development cycle, the “merge window” is said to be open. At that time, code which is deemed to be sufficiently stable (and which is accepted by the development community) is merged into the mainline kernel. The bulk of changes for a new development cycle (and all of the major changes) will be merged during this time, at a rate approaching 1,000 changes (“patches,” or “change sets”) per day.The merge window lasts for approximately two weeks. At the end of this time, Linus Torvalds will declare that the window is closed and release the first of the “rc” kernels. For the kernel which is destined to be 2.6.40, for example, the release which happens at the end of the merge window will be called 2.6.40-rc1. The -rc1 release is the signal that the time to merge new features has passed, and that the time to stabilize the next kernel has begun.

Over the next six to ten weeks, only patches which fix problems should be submitted to the mainline. On occasion a more significant change will be allowed, but such occasions are rare; developers who try to merge new features outside of the merge window tend to get an unfriendly reception. As a general rule, if you miss the merge window for a given feature, the best thing to do is to wait for the next development cycle.As fixes make their way into the mainline, the patch rate will slow over time. Linus releases new -rc kernels about once a week; a normal series will get up to somewhere between -rc6 and -rc9 before the kernel is considered to be sufficiently stable and the final 2.6.x release is made. At that point the whole process starts over again.

As an example, here is how the 2.6.38 development cycle went (all dates in 2011):

|  |  |
| --- | --- |
| January 4 | 2.6.37 stable release |
| January 18 | 2.6.38-rc1, merge window closes |
| January 21 | 2.6.38-rc2 |
| February 1 | 2.6.38-rc3 |
| February 7 | 2.6.38-rc4 |
| February 15 | 2.6.38-rc5 |
| February 21 | 2.6.38-rc6 |
| March 1 | 2.6.38-rc7 |
| March 7 | 2.6.38-rc8 |
| March 14 | 2.6.38 stable release |

How do the developers decide when to close the development cycle and create the stable release? The most significant metric used is the list of regressions from previous releases. No bugs are welcome, but those which break systems which worked in the past are considered to be especially serious. For this reason, patches which cause regressions are looked upon unfavorably and are quite likely to be reverted during the stabilization period.

The developers’ goal is to fix all known regressions before the stable release is made. In the real world, this kind of perfection is hard to achieve; there are just too many variables in a project of this size. There comes a point where delaying the final release just makes the problem worse; the pile of changes waiting for the next merge window will grow larger, creating even more regressions the next time around. So most 2.6.x kernels go out with a handful of known regressions though, hopefully, none of them are serious.

Once a stable release is made, its ongoing maintenance is passed off to the “stable team,” currently consisting of Greg Kroah-Hartman. The stable team will release occasional updates to the stable release using the 2.6.x.y numbering scheme. To be considered for an update release, a patch must (1) fix a significant bug, and (2) already be merged into the mainline for the next development kernel. Kernels will typically receive stable updates for a little more than one development cycle past their initial release. So, for example, the 2.6.36 kernel’s history looked like:

|  |  |
| --- | --- |
| October 10 | 2.6.36 stable release |
| November 22 | 2.6.36.1 |
| December 9 | 2.6.36.2 |
| January 7 | 2.6.36.3 |
| February 17 | 2.6.36.4 |

2.6.36.4 was the final stable update for the 2.6.36 release.

Some kernels are designated “long term” kernels; they will receive support for a longer period. As of this writing, the current long term kernels and their maintainers are:

|  |  |  |
| --- | --- | --- |
| 2.6.27 | Willy Tarreau | (Deep-frozen stable kernel) |
| 2.6.32 | Greg Kroah-Hartman |  |
| 2.6.35 | Andi Kleen | (Embedded flag kernel) |

The selection of a kernel for long-term support is purely a matter of a maintainer having the need and the time to maintain that release. There are no known plans for long-term support for any specific upcoming release.